Zero Curvature Initialization of Neural Networks

Public Domain - Sean O'Connor - 21 August 2021

Most neural networks are initialized with random weights, followed perhaps, by some form of pre-training. Unfortunately, randomness remains embedded in the network despite any amount of training. It can also be argued that random initialization slows training by creating extremely complicated decision boundaries that the training algorithm has to somehow repair.

Setting all the weights to zero initially is possible in some cases but ill suits the training algorithms commonly used. A potential solution can be found by looking at the curvature of activation functions and zero curvature initialization.

The Curvature of ReLU

$$f(x) = \begin{cases} x & \text{if, } x > 0 \\ 0 & \text{if, } x \leqslant 0 \end{cases}$$

Figure 1. The ReLU activation function.

Curvature is the overall amount by which a curve deviates from being a straight line. The curve for the ReLU activation function is shown in figure 2.

The first half of the plot is a horizonal line, which by itself has no curvature.

The second half of the plot is a 45-degree line, which again, by itself has no curvature. If the 45-degree line continued on indefinitely with no curves or bumps it would be a straight line with no curvature.

Overall though, ReLU has a rather drastic amount of curvature due to the switch in slope that happens at zero.

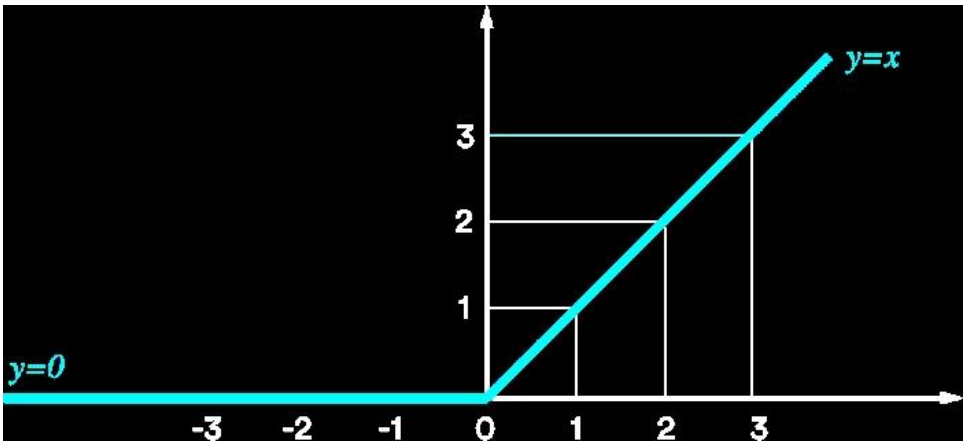


Figure 1. A plot of the ReLU function.

ReLU and PReLU

In slope-intercept form, the equation of the line is given by:

y = mx + b

Where m is the slope and b is the y intercept.

For ReLU the slope is zero and then switches to 1 for inputs greater than or equal to 0. The y intercept b is obviously 0.

ReLU is a fixed function that has no adjustable parameters.

In contrast the Parametric activation function PReLU in figure 3 takes one parameter. The parameter defines the slope response of the activation function when the input is less than zero. The slope when x is greater than or equal to zero remains at 1 and that graphs to the same 45-degree line as seen with ReLU.

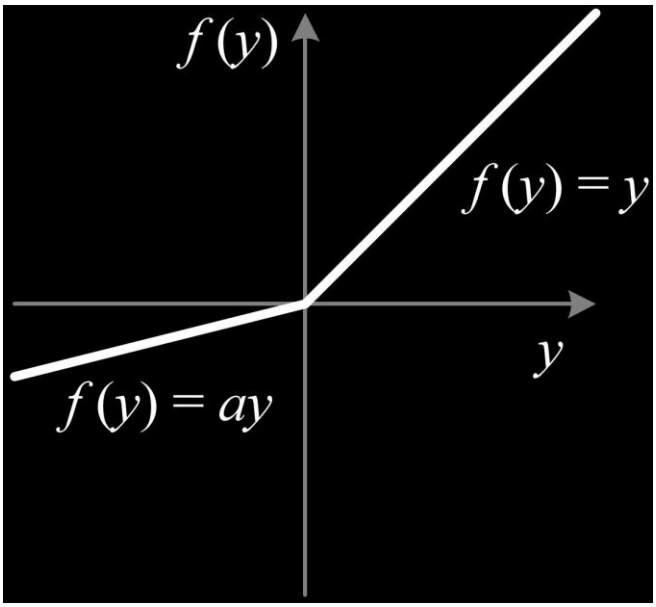


Figure 3. PReLU. Parametric ReLU, where a is the user defined slope.

Zero Curvature - First Option

Setting the free parameter of all the PReLU functions in a neural network to 1 gives the zero-curvature response as shown in figure 4. This allows zero curvature initialization. A question remains about how to initialize the weights.

You can set the weights randomly. Then you have random weighted sums interacting with zero curvature activation functions. Each layer is a simple linear mapping and the entire neural network is a composition of linear mappings. Basically, nothing much happens to data passing through the net except it is mixed around linearly a bit, in a way that is relatively easy to reverse.

In contrast the interaction in a ReLU neural network between the random initial weights and the ReLU activation function results drastic slope switching and high initial curvature for data passing through.

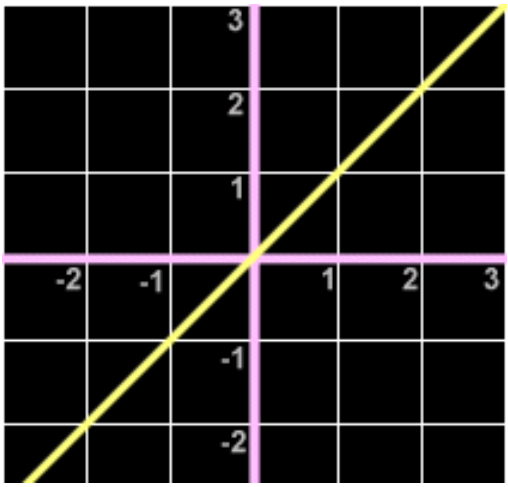


Figure 4. PReLU where the adjustable slope is set to 1 gives a zero curvature straight line.

Zero Curvature - Second Option.

Using PReLU in the first option removes the initial curvature and that makes training easier. It still leaves a lot of randomness in the system which makes a neural network's response and decision boundaries noisy and muddled. You could set some or all the weights initially to some constant and that will certainly help. There is a better way. You can embed a self-inverse fast transform into the weights of the neural network. Then the first layer of the network enacts the transform and a second layer will enact another transform unwinding the first layer's effect. Assuming the PReLU activation functions have been set to zero curvature then every 2 layers the output will be an exact copy of the input, as though nothing had happened.

The Walsh Hadamard transform is a good choice. Figure 5 shows the Hadamard matrix of order 8. For a neural network layer of width 8 you would copy the first row into the weights of the first neuron, the second row into the weights of the second neuron etc. Note that a constant scaling factor is necessary to make the Walsh Hadamard transform truly self-inverse.

$$\begin{bmatrix} +1 & +1 & +1 & +1 & +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 & +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 & +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 & +1 & -1 & -1 & +1 \\ +1 & +1 & +1 & +1 & -1 & -1 & -1 & -1 \\ +1 & -1 & +1 & -1 & -1 & +1 & -1 & +1 \\ +1 & +1 & -1 & -1 & -1 & -1 & +1 & +1 \\ +1 & -1 & -1 & +1 & -1 & +1 & +1 & -1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix}$$

Figure 5. The order 8 Hadamard matrix. Which forms a self-inverse transform when applied to a column vector.

Two Slope Parametric Activation Function

You can also consider using the Two Slope Parametric activation function as shown in figure 6. It allows substantially greater control than PReLU.

$$f(x) = \begin{cases} a.x & \text{if, } x > 0 \\ b.x & \text{if, } x \leqslant 0 \end{cases}$$

Figure 6. The Two Slope Parametric activation function.

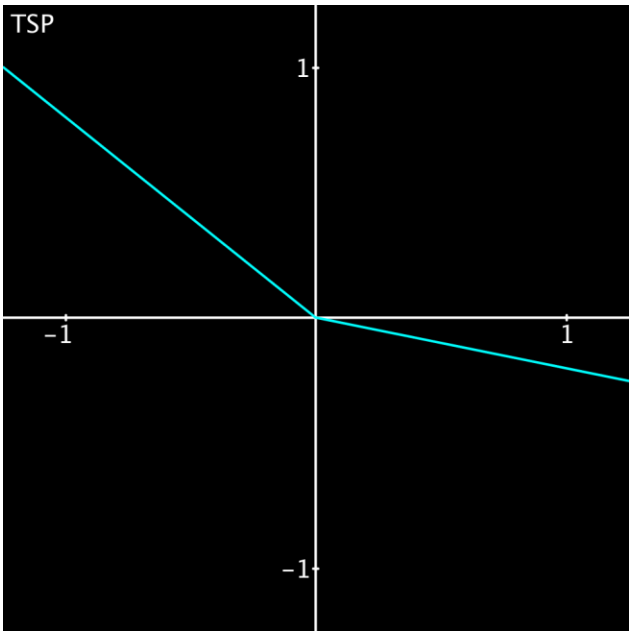
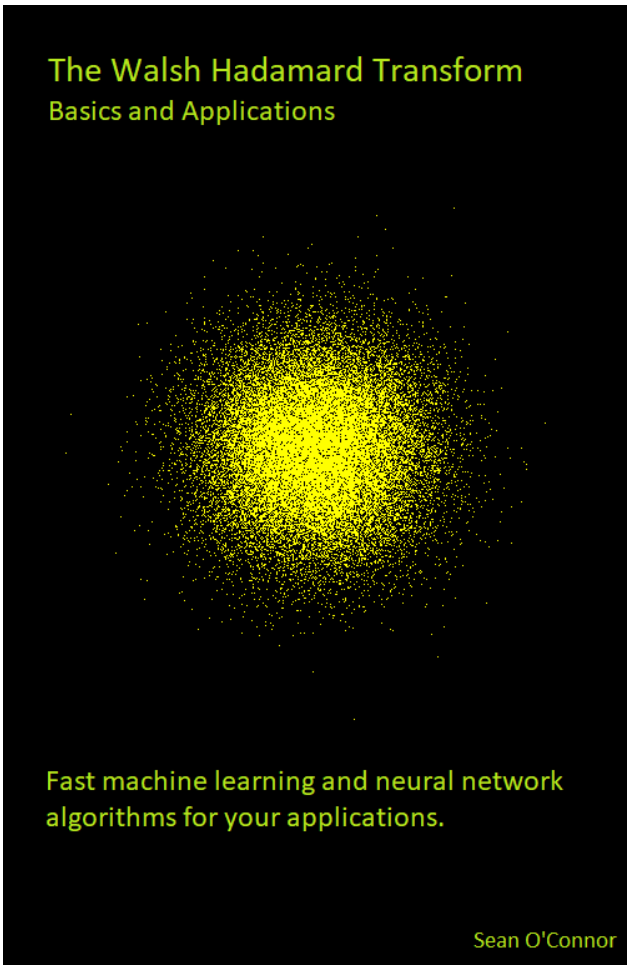


Figure 7. An example of the TSP activation function in action.



[The Walsh Hadamard Tranform Information Kit. Available Now!]

-